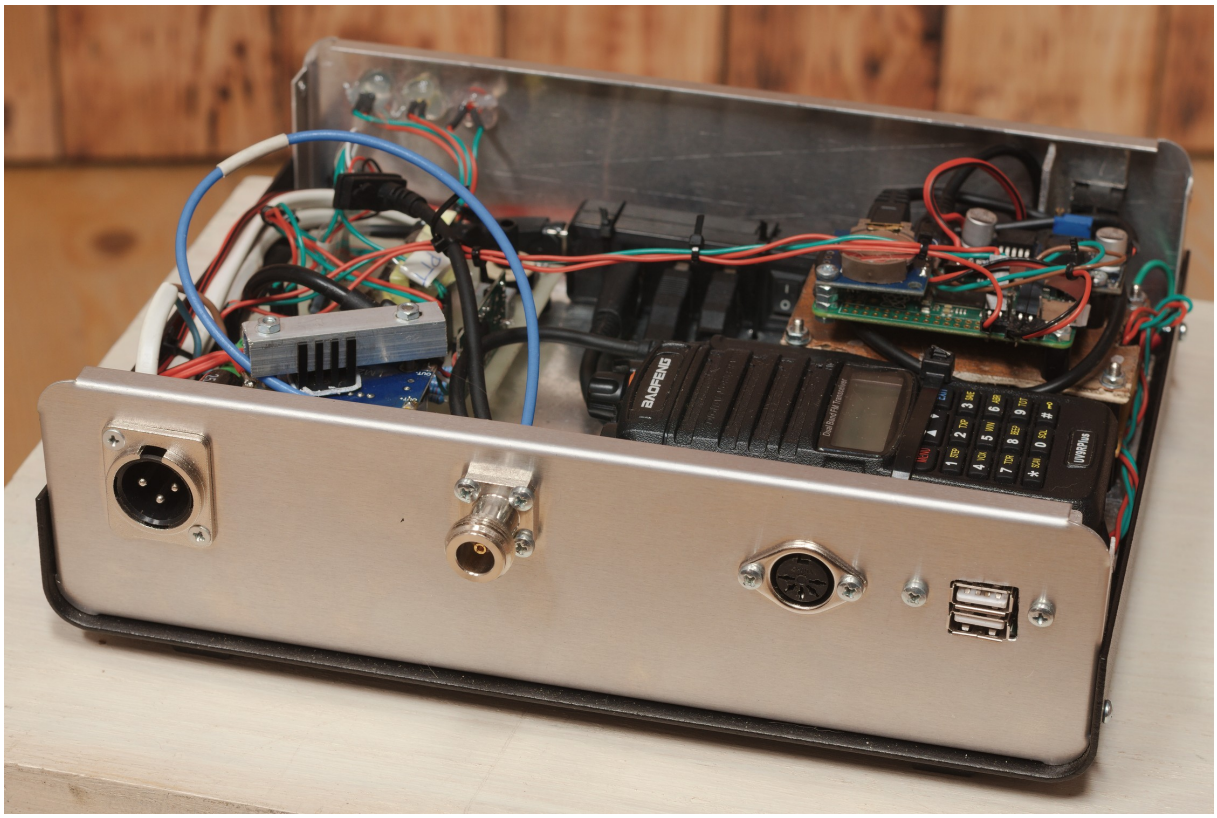
PE1RXF APRS Server

M.T. Konstapel

2021-12-30



Abstract

This document describes the build and use of an APRS server, designed for the 2 meter and 70 cm amateur radio bands. It is based around a Raspberry Pi Zero and has a build in radio as well as an option for connecting an external second radio. At can be configured as a digipeater (single, dual or cross band), an igate and - by means of some custom programs - can be used for sending and receiving APRS messages. It also can decode and process the custom telemetry data protocol from PE1RXF data logging devices. It is fully configurable by the command line and the most used functions are available via a web site.

1 An overview

1.1 Hardware

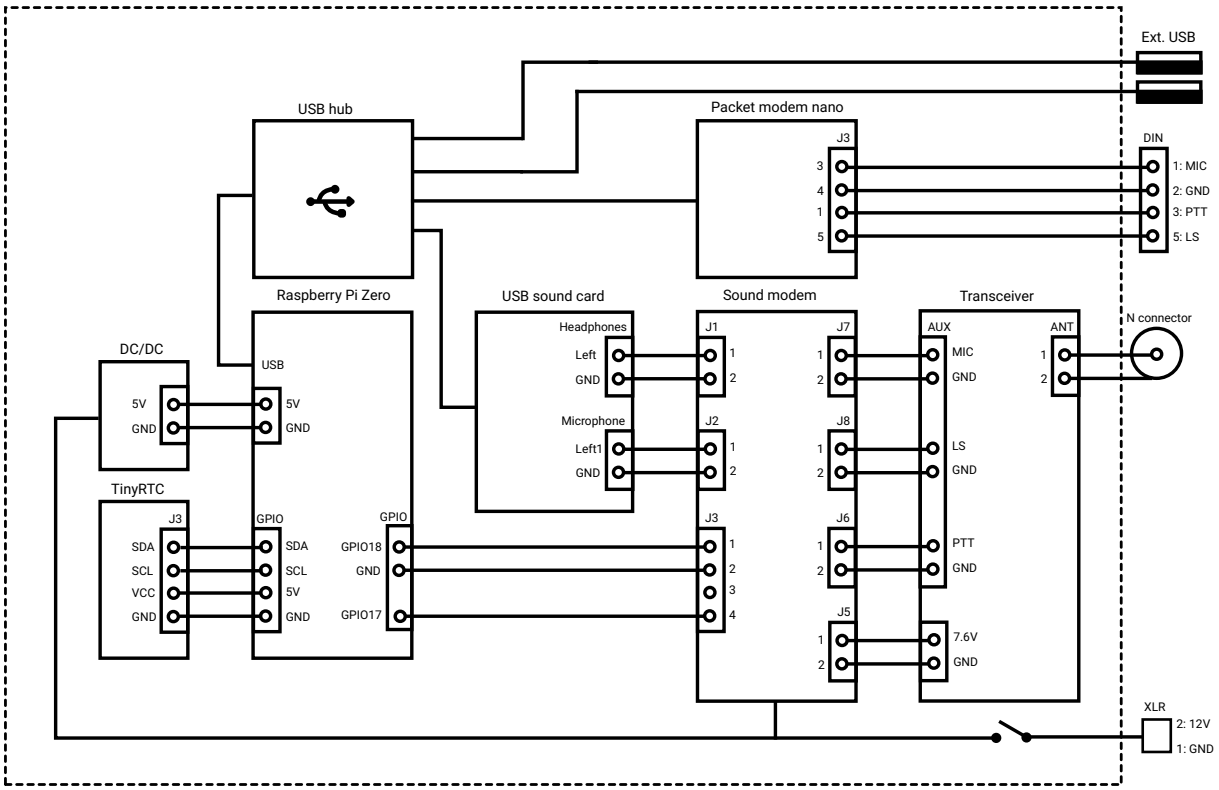


Figure 1: block diagram of the hardware

The heart of the device is a Raspberry Pi Zero. Connected to it via USB are two AFSK1200 modems. One a sound modem, connected to a Baofeng portable radio. The other an Arduino based KISS modem, wired to a 5 pin DIN connector on the back of the device. A second FM transceiver can be connected to this DIN plug. Both modems are galvanically isolated from the transceivers.

A real time clock keeps the time and two USB ports are available on the back of the device. These ports can be used for anything. For example: one can be used for connecting a WIFI or Ethernet adapter, the other for connecting a third KISS modem for APRS over LoRa.

The device is powered by 12 Volt.

1.2 Software

The Raspberry Pi Zero runs Raspberry Pi OS Bullseye. Both modems are available through the ax25 ports ax0 and ax1. The sound modem is handled by Direwolf and attached to port ax0. The packet modem nano is attached to port ax1. These two ax25 ports are then bind to aprx, which handles most of the APRS traffic.

Some functions are handled by the custom software. These include beaconing, generating acknowledges, sending and receiving message, generating some basic statistics and decoding the custom telemetry data.

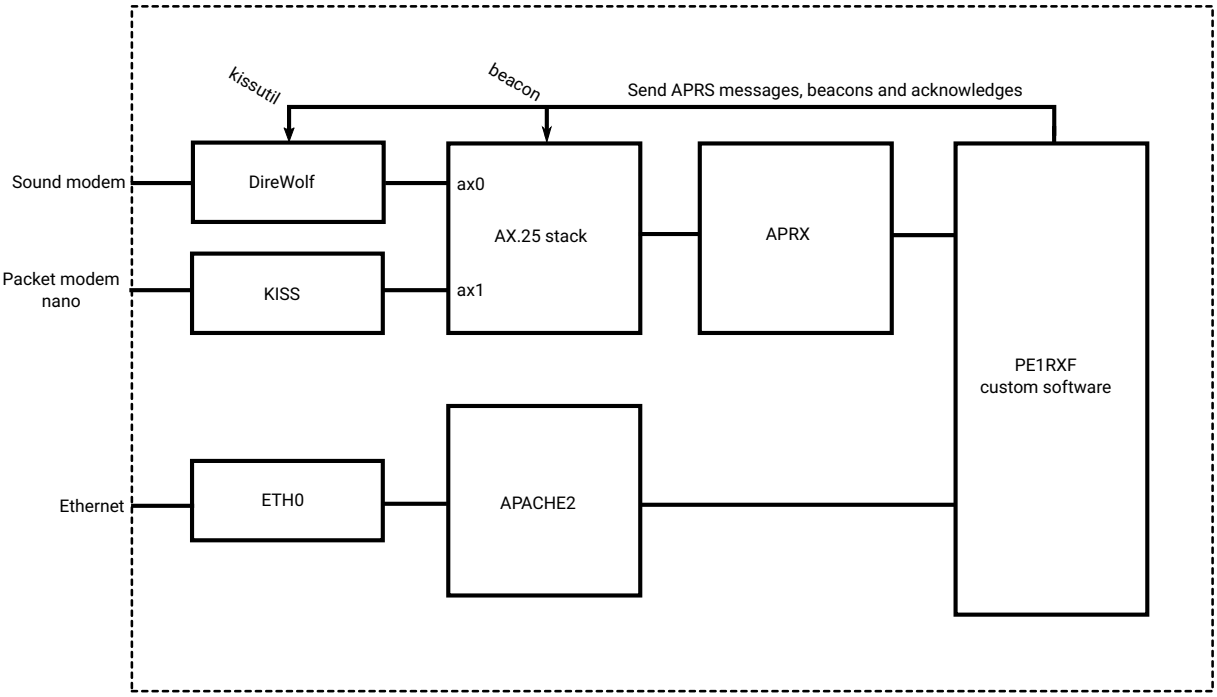


Figure 2: block diagram of the software

When connected to Ethernet or WiFi, most functions are available via the Apache web server and the whole system can be configured and maintained via SSH.

2 Hardware: a detailed view

2.1 Sound modem

The sound modem is constructed from an off the shelf USB sound card (figure 4a) and a custom radio interface (figure 3).

The USB sound card is a generic card with a microphone input and a headphones output. The output must me capable of driving a 600 Ohm load. In order to safe space, the plastic enclosure is removed and the pcb is directly soldered to the radio interface.

The radio interface provides galvanic isolation of the radio. This prevents ground loops. Because the radio is build in, this is not strictly necessary. But it is still good practice and it makes the interface more versatile. With RV1 and RV3, the receive and transmit sensitivity can be set.

The PTT circuit has a watchdog build around a 555 timer. With RV2, the watchdog time is set to approximately half a minute. This limits the transmit time of the transceiver to about 30 seconds.

An optocoupler isolates the CD (carrier detect) LED. Again, the isolation is not really needed here, but the interface was designed to be universal. With this design it is possible to replace the build in transceiver with an external transceiver, powered by a separate power supply, without having to worry about ground loops.

The watchdog and the status LEDs are powered by 7.6V from the DC/DC-converter. This odd voltage is chosen because the internal radio is also powered from this supply.

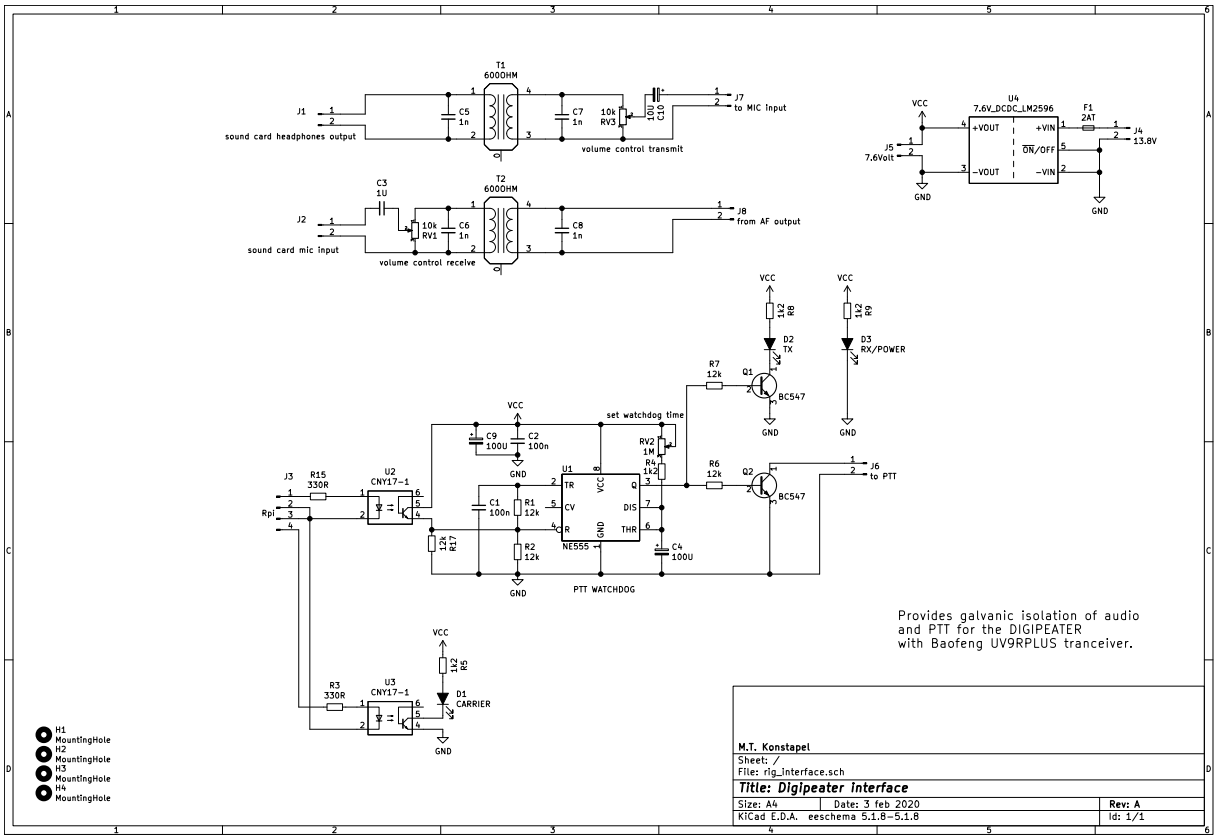


Figure 3: sound modem

2.2 Internal transceiver

(a) usb sound card (b) Baofeng UV-9Rplus

Figure 4: the peripherals

The sound modem is connected to the internal transceiver. This can be any random FM transceiver. I use a Baofeng UV9Rplus (figure 4b). It works surprisingly well given

the reputation of these cheap radios. But I do not dare to hook it up to a spectrum analyzer. You should use a reputable brand!

2.3 Packet modem nano

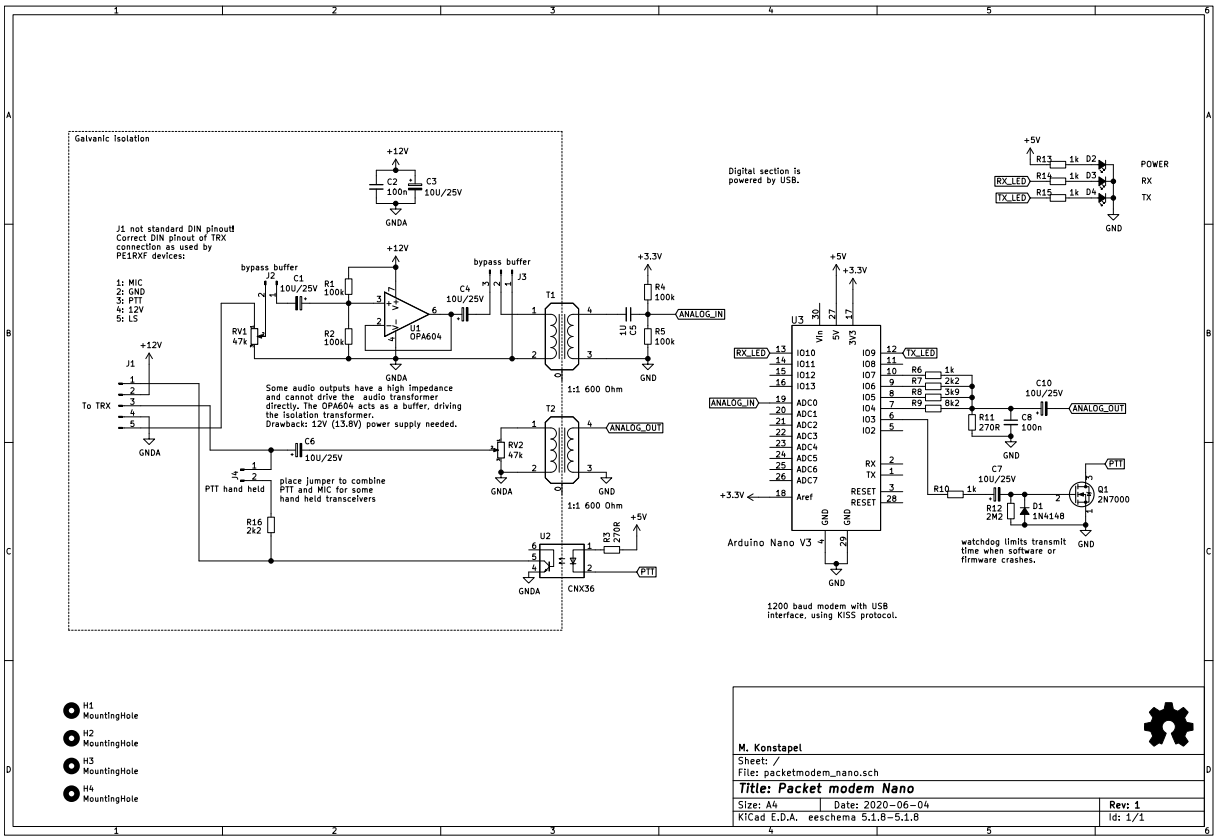


Figure 5: packet modem nano

When I started using packet radio in the early 90s, the only thing I could afford was a cheap baycom modem. And while it worked great, I always dreamed of having a real TNC. Now I finally could build one and it only costs me a mere twenty Euros. It is based on the MicroModem[1] designed by markqvist (figure 5).

The modem uses the standard KISS firmware from markqvist.

2.4 Real time clock

While you can hook up the device to the internet and have the luxury of the **N**etwork **T**ime **P**rotocol, it must also be able to keep time when not connected to the net. So a **R**eal **T**ime **C**lock or RTC is connected to the Raspberry Pi via I^2C. I used a TinyRTC (figure 6).

The Raspberry Pi GPIO pins work with 3.3 Volts levels and are not 5 Volt tolerant. The TinyRTC is powered by 5 Volts but works reliably with 3.3 Volt signal levels. By removing the pull up resistors on the TinyRTC, the two can safely communicate with each other.

[1] https://unsigned.io/micromodem/

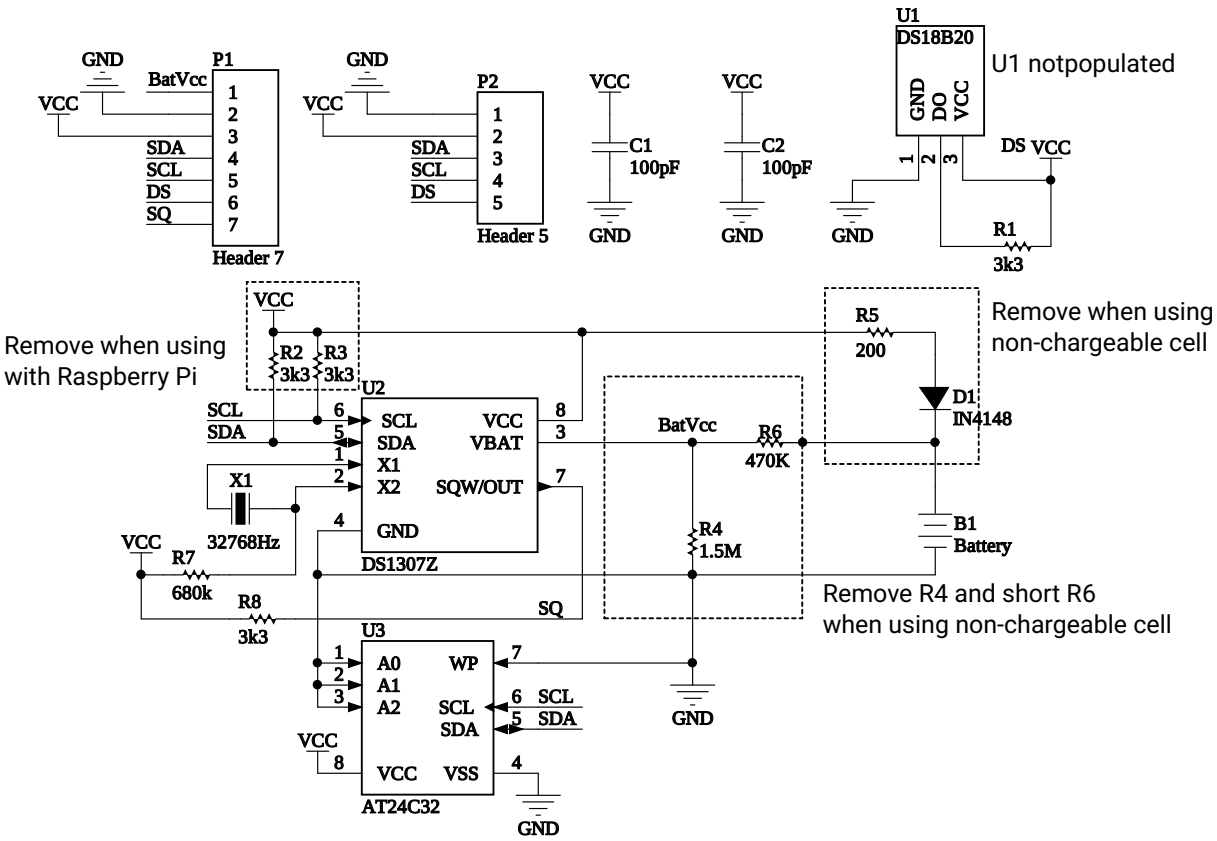


Figure 6: tinyRTC

2.5 Raspberry Pi Zero

The software runs on a Raspberry Pi Zero (figure 7). This tiny single board computer is more than capable of running the software. Even DireWolf, which is relatively power hungry, loads the cpu for only 15-20%. The Raspberry Pi Zero is powered by a DC/DC-converter (figure 8a) which converts the 12 Volt input voltage to the needed 5 Volts.

2.6 Connecting it all

The modems as well as the two external USB ports are connected to the Raspberry Pi Zero through a OTG USB-hub. Nothing fancy, just a cheap one (figure 8b).

Figure 9 shows the guts of the machine. The RTC, Raspberry Pi Zero and the DC/DC-converter are located at the top right. Hidden from sight just below that is the packet modem nano. A photograph of the modem is shown in figure 10.

The sound modem sits on the left. The usb sound card itself is removed from the plastic housing and soldered to this board at a 90 degrees angle. You have to look hard to see it, but it is there.

The wiring diagram can be found in figure 1.

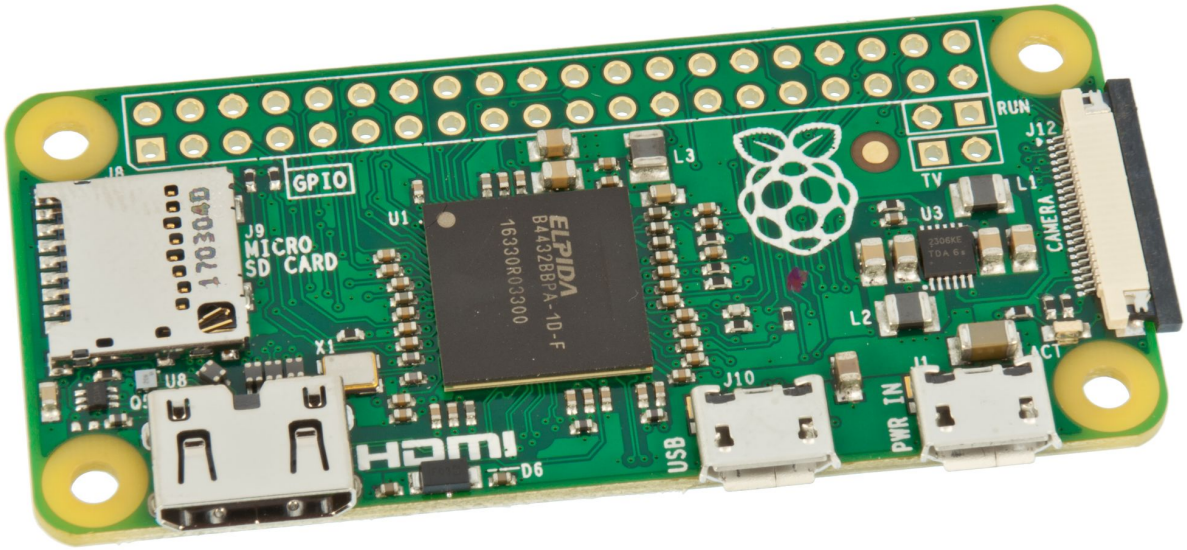
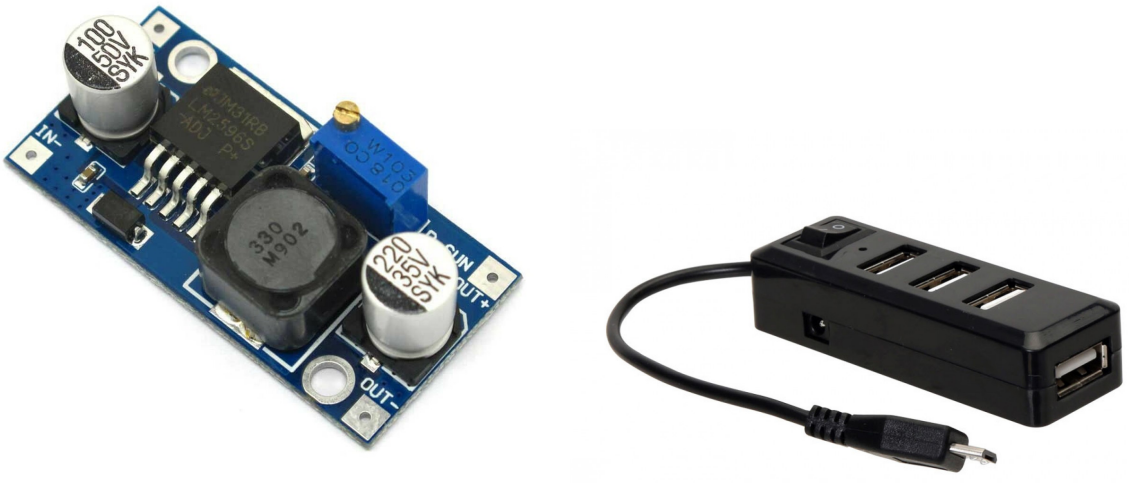


Figure 7: Raspberry Pi Zero



(a) DC/DC-converter (b) USB hub

Figure 8: the peripherals

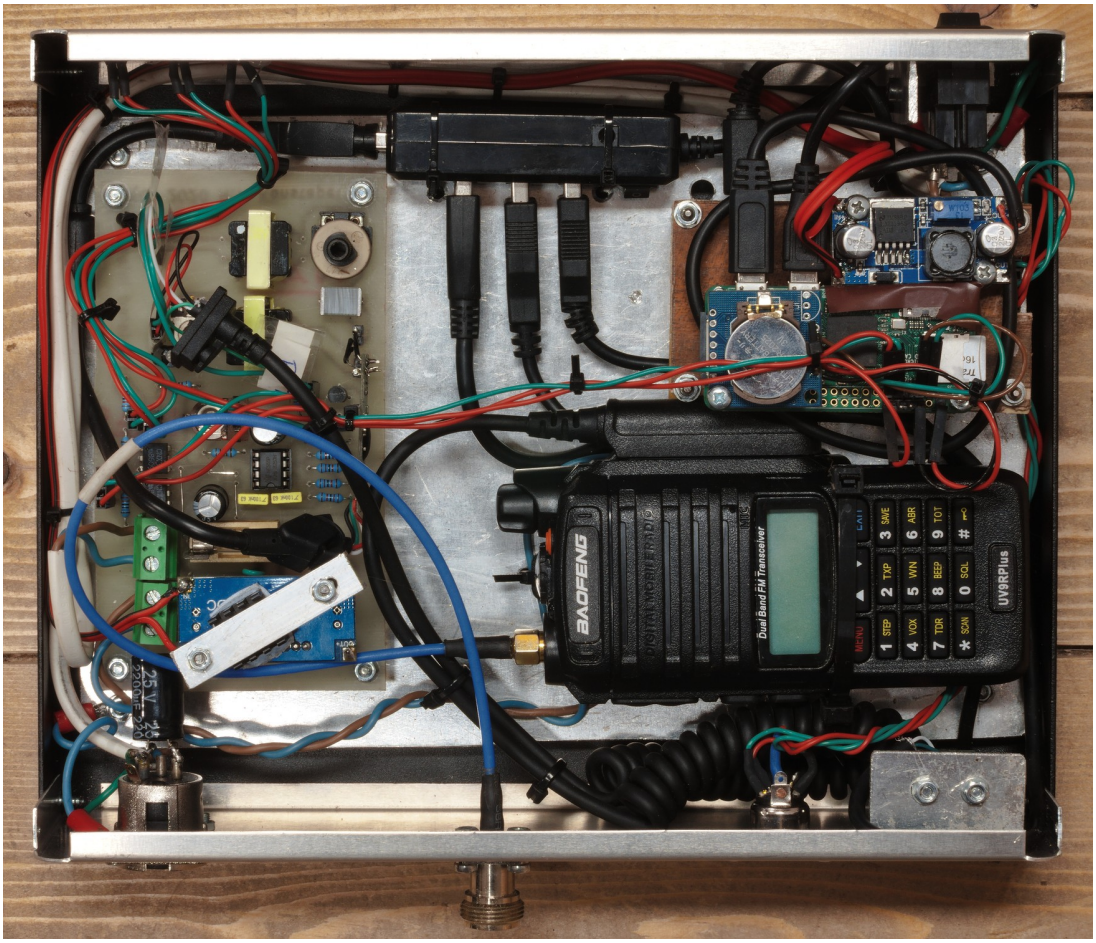


Figure 9: digipeater from above

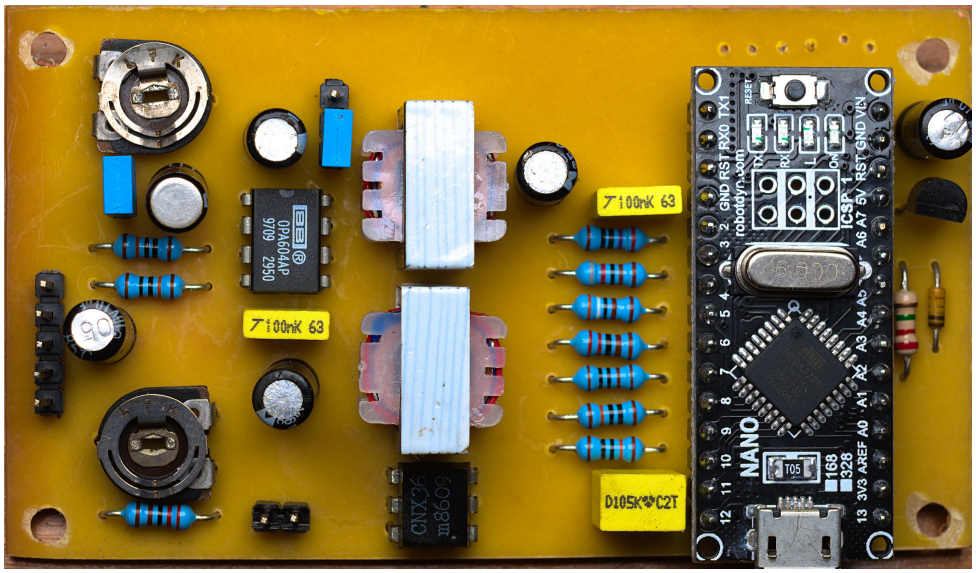


Figure 10: packet modem nano

3 Software: a detailed view

3.1 Operating system

The operating system used is Raspberry Pi OS Lite bullseye. The custom software from PE1RXF comes with the file *installation_pe1rxf-aprs-server.txt* which describes in detail how to set it up for this project.

3.2 Dire Wolf and kissattach

The sound modem is processed by Dire Wolf. Via kissattach it is attached to the AX.25 stack as port ax0. The packet modem nano, which is a KISS modem, is also attached to the AX.25 stack via kissattach. It is available as ax1.

Dire Wolf is the most processor intensive program of the digipeater, but still uses only about 15-20% cpu time.

3.3 AX.25 stack

The file /etc/ax25/axports defines the AX.25 ports ax0 and ax1. The software runs quietly in the background, doing its job.

3.4 APRX

The main functionality of the digipeater is performed by APRX. This program is widely used and performs very good on limited hardware.

3.5 PE1RXF custom software

Without it the digipeater would just be that: a digipeater. This software makes is more user friendly: via a web interface it is possible to send and receive messages, set beacons for each channel and see what calls were heard the last 24 hours. It also processes the custom APRS telemetry data form PE1RXF devices, presenting the data in nice graphs.

Settings can be altered in the main configuration file *ham/aprs_utils/pe1rxf-aprs-server.cfg*. If a parameter is not set, the defaults in file *ham/aprs_utils/pe1rxf-aprs-server.cfg.defaults* are used.

3.5.1 Utilities

Most features are available as individual small programs in the *ham/aprs_utils/* directory. These programs are used by the web interface and the main loop, but some can also be run from the command line.

set_beacon.sh: Set interval of APRS beacons in crontab.

```
Syntax: set_beacon.sh [-i <interface>|-t <time>|-c <config-file>]
Options:
-i      AX.25 interface (ax0 or ax1)
-t      Time interval in minutes (0-60), 0 disables beacon
-c      Read from config file instead of command line arguments
```

send_message.sh: Send APRS message.

```
Syntax: set_beacon.sh [-i <interface>|-c <call>|-p <path>|-m <message>]
Options:
-i      AX.25 interface (ax0 or ax1)
-c      Destination call
-p      Path: 0=none, WIDE2-1, WIDE2-2, WIDE3-3 or call of digipeater
-m      Message to send.
```

send_beacon_internal_radio.sh: Send the beacon on ax0 as defined in the configuration file.

send_beacon_external_radio.sh: Send the beacon on ax1 as defined in the configuration file.

3.5.2 Initialization and main loop

The custom software is started by calling *ham/start_aprs_server.sh* at boot via the crontab of the user who owns the program. This is described in the installation file. From here all the drivers are installed and the KISS-ports are attached to the AX.25 stack. The APRS data from APRX is redirected to the custom software and the main loop is started.

3.5.3 Web interface

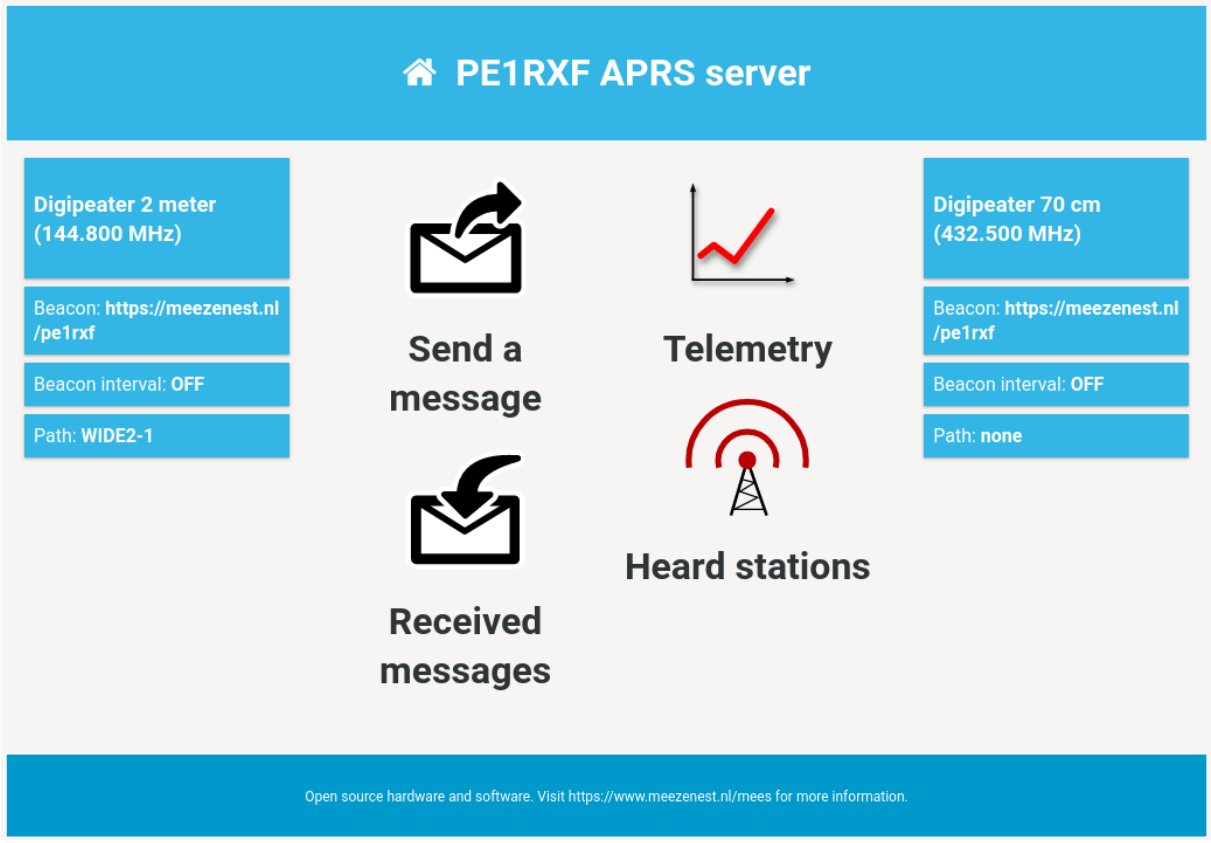


Figure 11: web interface: main page

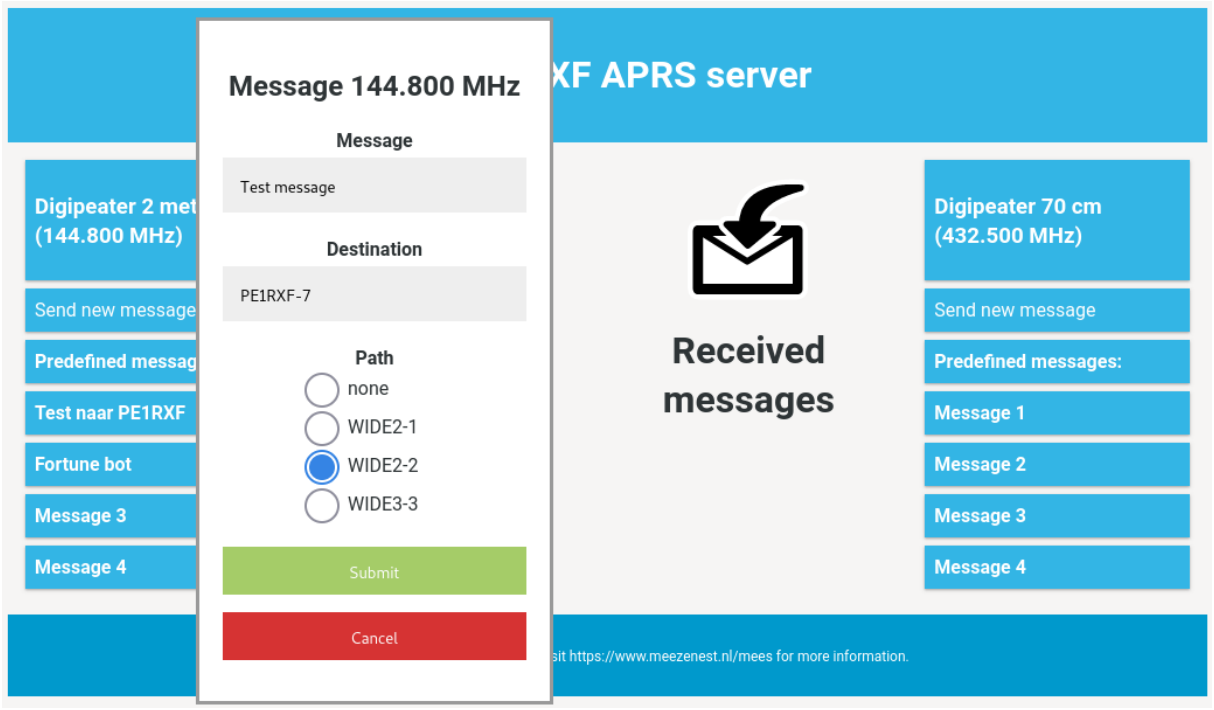


Figure 12: web interface: sending message

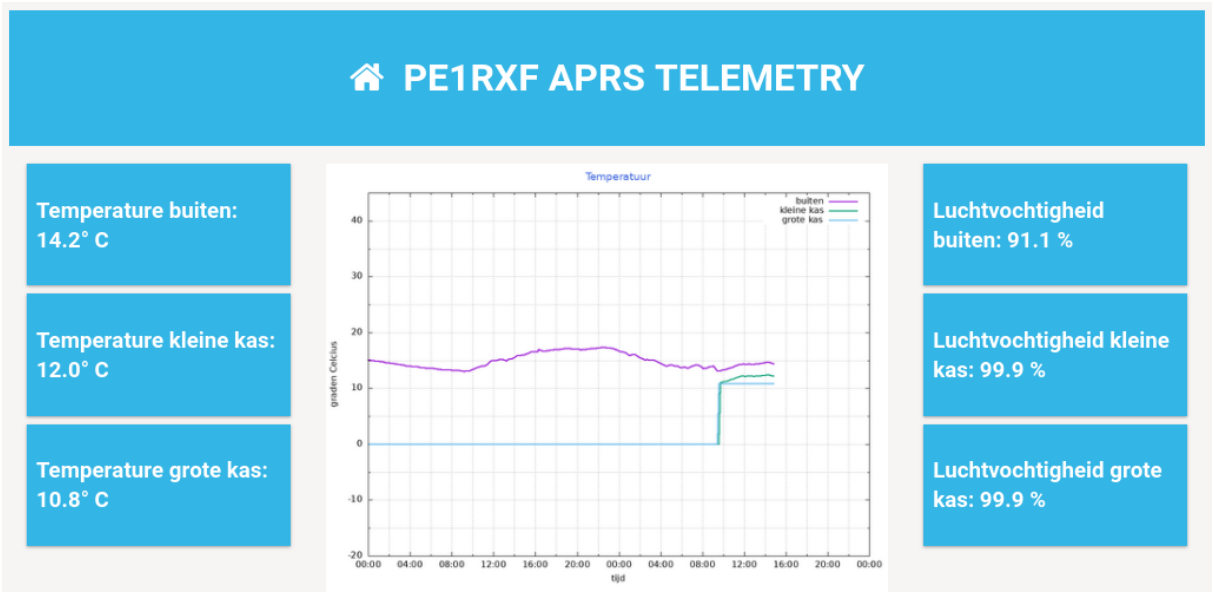


Figure 13: web interface: telemetry

A web site is used as a user interface. The pages are served by Apache2. In order to reach the server, a network adapter must be installed on one of the two external USB ports. An Ethernet adapter should work out of the box. A WiFi adapter must be configured by the user. The web page is available via http at port 80.

3.5.4 PE1RXF telemetry

This custom protocol for telemetry is described in detail on my website. It can be found at: *https://www.meezenest.nl/mees/aprs_telemetry.html*

4 Open source software and hardware

All the design files are available on my website: *https://www.meezenest.nl/mees*

The custom software is available at my git repositories: *https://git.meezenest.nl/*